10

System Administration: Core Concepts

In This Chapter

Objectives

After reading this chapter you should be able to:

- Explain the need for and the responsibility of a privileged user (**root**)
- Gain **root** privileges using su and sudo
- Describe the startup sequence using systemd
- Manage which services start at boot time
- List four characteristics of a well-maintained system
- Start and stop services on a running system
- Boot into single-user mode for system maintenance
- Shut down a running system
- Secure a system by applying updates, monitoring logs, and controlling access to files using SELinux, setuid permission, and PAM
- Use system administration tools to monitor and maintain the system
- List common steps for installing, configuring, and securing a server
- Configure a system using a static IP address or using DHCP

The job of a system administrator is to keep one or more systems in a useful and convenient state for users. On a Linux system, the administrator and user might both be you, with you and the computer being separated by only a few feet. Alternately, the system administrator might be halfway around the world, supporting a network of systems, with you being one of thousands of users. Or a system administrator can be one person who works part-time taking care of a system and perhaps is also a user of the system. In some cases several administrators might work together full-time to keep many systems running.

A well-maintained system:

- Runs quickly enough so users do not get frustrated waiting for the system to respond or complete a task
- Has enough storage to accommodate the reasonable needs of users
- Provides a working environment appropriate to each user's abilities and requirements
- Is secure from malicious and accidental acts altering its performance or compromising the security of the data it holds and exchanges with other systems
- Is backed up regularly, with recently backed-up files readily available to users
- Has recent copies of the software that users need to get their jobs done
- Is easier to administer than a poorly maintained system

In addition, a system administrator should be available to help users with all types of system-related problems—from logging in to obtaining and installing software updates to tracking down and fixing obscure network issues.

Part III of this book breaks system administration into nine chapters.

- Chapter 9 covers bash (Bourne Again Shell) to a depth that you can use it interactively to administer a system and begin to understand complex administration shell scripts.
- Chapter 10 covers the core concepts of system administration, including working with **root** (Superuser) privileges, system operation, configuration tools and other useful utilities, general information about setting up and securing a server (including a section on DHCP), and PAM.
- Chapter 11 covers files, directories, and filesystems from an administrator's point of view.
- Chapter 12 covers installing software on the system, including the use of yum, BitTorrent, and curl.
- Chapter 13 discusses how to set up local and remote printers that use the CUPS printing system.

- Chapter 14 explains how to rebuild the Linux kernel and work with GRUB, the Linux boot loader.
- Chapter 15 covers additional system administrator tasks and tools, including setting up users and groups, backing up files, scheduling tasks, printing system reports, and general problem solving.
- Chapter 16 goes into detail about how to set up a LAN, including setting up and configuring network hardware and configuring software.
- Chapter 17 describes how to set up virtual machines locally (gnome-boxes, KVM/QEMU, and VMware) and in the cloud (AWS).

Because Linux is readily configurable and runs on a wide variety of platforms, this chapter cannot discuss every system configuration or every action you might have to take as a system administrator. Instead, this chapter seeks to familiarize you with the concepts you need to understand and the tools you will use to maintain a Linux system. Where it is not possible to go into depth about a subject, the chapter provides references to other sources.

This chapter assumes you are familiar with the following terms:

block device (page 1239)
daemon (page 1245)
device (page 1246)
device filename (page 1247)
disk partition (page 1247)
environment (page 1249)
filesystem (page 1250)
fork (page 1250)
kernel (page 1257)
login shell (page 1259)
mount (page 1261)
process (page 1267)
root filesystem (page 1271)
runlevel (page 1271)
signal (page 1273)
spawn (page 1274)
system console (page 1276)
X server (page 1281)

If there is a problem, check the log files

tip If something on the system is not working as expected, check the log files in **/var/log**. This directory holds many files and subdirectories. If you cannot connect to a server, also check the log files on the server.

If something does not work, see if the problem is caused by SELinux

tip If a server or other system software does not work properly, especially if it displays a permissions-related error message, the problem might lie with SELinux. To see whether SELinux is the cause of the problem, put SELinux in permissive mode and run the software again. If the problem goes away, you need to modify the SELinux policy. Remember to turn SELinux back on. See the tip on page 473 and refer to "Setting the Targeted Policy using system-config-selinux" on page 475.

If you cannot access a remote system, check the firewall

tip If a server does not appear to work or you cannot access a remote system, make sure the firewall is not the problem. On a non-production system, use systemctl to turn the firewall off (page 900) and see if the problem goes away. Then turn the firewall back on and open only the necessary port using firewall-cmd (page 906).

LE RUNNING COMMANDS WITH root PRIVILEGES

Some commands can damage the filesystem or crash the operating system. Other commands can invade users' privacy or make the system less secure. To keep a Linux system up and running as well as secure, most systems are configured not to permit ordinary users to execute some commands and access certain files. Linux provides several ways for a trusted user to execute these commands and access these files. A user running with these privileges is sometimes referred to as an *administrator*, *privileged user*, or *Superuser*.

Administrator

security During installation (page 68) and whenever you add or modify a user (pages 112 and 598), you have the opportunity to specify that user as an *administrator.* Two characteristics give an administrator special privileges. First, an administrator is a member of the **wheel** group (page 429) and as such can use sudo to authenticate using her password; she does not need to know the **root** password. Second, **polkit** (www.freedesktop.org/wiki/Software/polkit) is set up so that an administrator can do some kinds of administrative work on the desktop (e.g., updating software and adding a printer) without needing to enter a password.

LE THE SPECIAL POWERS OF A PRIVILEGED USER

root privileges A user running with **root** privileges has the following powers—and more.

- Some commands, such as those that add new users, partition hard drives, and change system configuration, can be executed only by a user working with **root** privileges. Such a user can configure tools, such as sudo, to give specific users permission to perform tasks that are normally reserved for a user running with **root** privileges.
- Read, write, and execute file access and directory access permissions do not affect a user with **root** privileges. A user with **root** privileges can read from, write to, and execute all files, as well as examine and work in all directories.

  Exceptions to these privileges exist. For example, SELinux mandatory access can be configured to limit **root** access to a file. Also, a user working with **root** privileges cannot make sense of an encrypted file without possessing the key.
- Some restrictions and safeguards that are built into some commands do not apply to a user with **root** privileges. For example, a user with **root** privileges can change any user's password without knowing the old password.

Console security

security Linux is not secure from a person who has physical access to the computer. Additional security measures, such as setting boot loader and BIOS passwords, can help secure the computer. However, if someone has physical access to the hardware, as system console users typically do, it is very difficult to secure a system from that user.

prompt When you are running with **root** privileges in a command-line environment, by convention the shell displays a special prompt to remind you of your status. By default, this prompt is (or ends with) a hashmark (#).

Least privilege

caution When you are working on any computer system, but especially when you are working as the system administrator (working with **root** privileges), perform any task while using the least privilege possible. When you can perform a task logged in as an ordinary user, do so. When you must run a command with **root** privileges, do as much as you can as an ordinary user, log in or use su or sudo so you have **root** privileges, complete the part of the task that has to be done with **root** privileges, and revert to being an ordinary user as soon as you can. Because you are more likely to make a mistake when you are rushing, this concept becomes even more important when you have less time to apply it.

LE GAINING root PRIVILEGES

Classically a user gained **root** privileges by logging in as the user named **root** or by giving an su (substitute user) command and providing the **root** password. More recently the use of sudo has taken over this classic technique of gaining **root** privileges. With sudo, the user logs in as herself, gives an **sudo** command, and provides her own password (not the **root** password) to gain **root** privileges. "Advantages of sudo" on page 428 discusses some of the advantages of sudo.

Graphical environment When an ordinary user executes a privileged command in a graphical environment, the system prompts for the **root** password or the user's password, depending on how the system is set up.

Some distributions lock the **root** account by not assigning a **root** password. On these systems, you cannot gain **root** privileges using a technique that requires you to supply the **root** password unless you unlock the **root** account as explained on page 438. Fedora/RHEL assigns a **root** password when the system is installed, so you can use these techniques from the start.

There is a root account, but no root password

tip As installed, some systems (not Fedora/RHEL) lock the **root** account by not providing a **root** password. This setup prevents anyone from logging in to the **root** account (except when you bring the system up in single-user mode [page 450]). There is, however, a **root** account (a user with the username **root**—look at the first line in **/etc/passwd**). This account/user owns files (give the command **ls –l /usr/bin**) and runs processes (give the command **ps –ef** and look at the left column of the output). The **root** account is critical to the functioning of a Linux system.

When properly set up, the sudo utility enables you to run a command as though it had been run by a user logged in as **root**. This book uses the phrase **working** (or **run**) **with root privileges** to emphasize that, although you might not be logged in as **root**, when you use su or sudo you have the powers of the **root** user.

The following list describes some of the ways you can gain or grant **root** privileges. Some of these techniques depend on you supplying the password for the **root**

account. Again, if the **root** account is locked, you cannot use these techniques unless you unlock the **root** account (set up a **root** password), as explained on page 438. Other techniques depend on the **sudoers** file being set up to allow you to gain **root** privileges (page 433). If this file is not set up in this manner, you cannot use these techniques.

- When you bring the system up in single-user/rescue mode (page 450), you log in as the user named **root**.
- You can give an su (substitute user) command while you are logged in as yourself. When you then provide the **root** password, you will be running with **root** privileges. See page 425.
- The sudo utility allows specified users to run selected commands with **root** privileges while they are logged in as themselves. You can set up sudo to allow certain users to perform specific tasks that require **root** privileges without granting these users systemwide **root** privileges. See page 428.
- Once the system is up and running in multiuser mode (page 452), you can log in as **root**. When you then supply the **root** password, you will be running with **root** privileges.
- Some programs ask for a password (either your password or the **root** password, depending on the command and the configuration of the system) when they start or when you ask them to perform certain tasks. When you provide a password, the program runs with **root** privileges. You stop running as a privileged user when you quit using the program. This setup keeps you from remaining logged in with **root** privileges when you do not need or intend to be.

LE+ Setuid file

- Any user can create a *setuid* (set user ID) file. Setuid programs run on behalf of the owner of the file and have all the access privileges the owner has. While you are working with **root** privileges, you can change the permissions of a file owned by **root** to setuid. When an ordinary user executes a file that is owned by **root** and has setuid permissions, the program has *effective **root** privileges*. In other words, the program can do anything a program running with **root** privileges can do that the program normally does. The user's privileges do not change. When the program finishes running, all user privileges are as they were before the program started. Setuid programs owned by **root** are both extremely powerful and extremely dangerous to system security, which is why a system contains very few of them. Examples of setuid programs that are owned by **root** include passwd, at, and crontab. For more information refer to "Setuid and Setgid Permissions" on page 196 and "Real UID Versus Effective UID" (next).

Logging in

The **/etc/securetty** file controls which terminals (ttys) a user can log in on as **root**.

Using the **/etc/security/access.conf** file, PAM controls the who, when, and how of logging in. Initially this file contains only comments. See page 476, the comments in the file, and the **access.conf** man page for details.

root-owned setuid programs are extremely dangerous

security Because **root**-owned setuid programs allow someone who does not know the **root** password and cannot use sudo to gain **root** privileges, they are tempting targets for a malicious user. Also, programming errors that make normal programs crash can become **root** exploits in setuid programs. A system should have as few of these programs as possible. You can disable setuid programs at the filesystem level by mounting a filesystem with the **nosuid** option (page 522). You can also use SELinux (page 472) to disable setuid programs. See page 458 for a find command that lists all setuid files on the local system. Future releases of Fedora/RHEL will remove most setuid files; see fedoraproject.org/wiki/Features/RemoveSETUID.

Do not allow root access over the Internet

security Prohibiting a user from logging in as **root** over a network is the default policy of Fedora/RHEL. The **/etc/securetty** file must contain the names of all devices you want a user to be able to log in on as **root**.

You can, however, log in as **root** over a network using ssh (page 685). As shipped by Fedora/RHEL, PAM configuration for ssh does not call the modules that use the **securetty** or **access.conf** configuration files. Also, in **/etc/ssh/sshd_config**, Fedora/RHEL sets **PermitRootLogin** to **yes** (it is set by default) to permit **root** to log in using ssh (page 705). For a more secure system, change **PermitRootLogin** to **no**.

LE REAL UID VERSUS EFFECTIVE UID

UID and username

A UID (user ID) is the number the system associates with a username. UID 0 is typically associated with the username **root**. To speed things up, the kernel keeps track of a user by UID, not username. Most utilities display the associated username in place of a UID.

The kernel associates two UIDs with each process: a *real UID* and an *effective UID*. The third column of the **/etc/passwd** file (or NIS/LDAP) specifies your real UID. When you log in, your real UID is associated with the process running the login shell. Because you have done nothing to change it, the effective UID associated with the process running the login shell is the same as the real UID associated with that process.

process privilege

The kernel determines which privileges a process has based on that process' effective UID. For example, when a process asks to open a file or execute a program, the kernel looks at the effective UID of the process to determine whether it is allowed to do so. When a user runs a setuid program (page 196), the program runs with the UID of the owner of the program, not the user running the program.

terminology: root privileges

When this book uses the phrase **run with root privileges** or **gain root privileges,** it means run with an effective UID of 0 (**root**).

LE+ USING su TO GAIN root PRIVILEGES

When you install Fedora/RHEL, you assign a password to the **root** account. Thus you can use su to gain **root** privileges without any further setup.

The su (substitute user) utility can spawn a shell or execute a program with the identity and privileges (effective UID) of a specified user, including **root:**

- Follow **su** on the command line with the name of a user; if you are working with **root** privileges or if you know the user's password, the newly spawned shell will take on the identity (effective UID) of that user.
- When you give an su command without an argument, su defaults to spawning a shell with **root** privileges (you have to supply the **root** password). That shell runs with an effective UID of 0 (**root**).

SPAWNING A root SHELL

When you give an su command to work with **root** privileges, su spawns a new shell, which displays the **#** prompt. That shell runs with an effective UID of 0 (**root**). You can return to your normal status (and your former shell and prompt) by terminating this shell: Press CONTROL-D or give an **exit** command.

LE who, whoami The who utility, when called with the arguments **am i**, displays the real UID (translated to a username) of the process that called it. The whoami utility displays the effective UID (translated to a username) of the process that called it. As the following example shows, the su utility (the same is true for sudo) changes the effective UID of the process it spawns but leaves the real UID unchanged.

```
$ who am i
sam      pts/2        2013-06-12 13:43 (192.168.206.1)
$ whoami
sam
$ su
Password:
# who am i
sam      pts/2        2013-06-12 13:43 (192.168.206.1)
# whoami
root
```

LE id Giving an su command without any arguments changes your effective user and group IDs but makes minimal changes to the environment. For example, **PATH** has the same value as it did before you gave the su command. The id utility displays the effective UID and GID of the process that called it as well as the groups the process is associated with. In the following example, the information that starts with **context** pertains to SELinux:

```
$ pwd
/home/sam
$ echo $PATH
/usr/local/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/home/sam/.local/bin:/home/sam/bin
$ id
uid=1000(sam) gid=1400(pubs) groups=1400(pubs),10(wheel) context=unconfined_u: ...
$ su
Password:
# pwd
/home/sam
# echo $PATH
/usr/local/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/home/sam/.local/bin:/home/sam/bin
```

```
# id
uid=0(root) gid=0(root) groups=0(root) context=unconfined_u: ...
# exit
exit
$
```

When you give the command **su –** (you can use **–l** or **––login** in place of the hyphen), su provides a **root** login shell: It is as though you logged in as **root**. Not only do the shell's effective user and group IDs match those of **root**, but the environment is the same as when you log in as **root**. The login shell executes the appropriate startup files (page 329) before displaying a prompt, and the working directory is set to what it would be if you had logged in as **root** (**/root**). **PATH** is also set as though you had logged in as **root**. However, as who shows, the real UID of the new process is not changed from that of the parent process.

```
$ su -
Password:
# pwd
/root
# echo $PATH
/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin:/root/bin
# who am i
sam      pts/2        2013-06-12 13:43 (192.168.206.1)
```

EXECUTING A SINGLE COMMAND

You can use su with the **–c** option to run a command with **root** privileges, returning to the original shell when the command finishes executing. In the following example, Sam tries to display the **/etc/shadow** file while working as himself, a nonprivileged user. The cat utility displays an error message. When he uses su to run cat to display the file, su prompts him for a password, he responds with the **root** password, and the command succeeds. The quotation marks are necessary because **su –c** takes the command it is to execute as a single argument.

```
$ cat /etc/shadow
cat: /etc/shadow: Permission denied

$ su -c 'cat /etc/shadow'
Password:
root:$6$il96HvSfmvep.m2F$2RI1LZ ... fYc3wYZFQ/:15861:0:99999:7:::
bin:*:15839:0:99999:7:::
daemon:*:15839:0:99999:7:::
adm:*:15839:0:99999:7:::
...
```

The next example first shows that Sam is not permitted to kill (page 465) a process. With the use of **su –c** and the **root** password, however, Sam is working with **root** privileges and is permitted to kill the process.

```
$ kill -15 4982
-bash: kill: (4982) - Operation not permitted
$ su -c "kill -15 4982"
Password:
$
```

The final example combines the **–** and **–c** options to show how to run a single command with **root** privileges in the **root** environment:

```
$ su -c pwd
Password:
/home/sam
$ su - -c pwd
Password:
/root
```

Root privileges, PATH, and security

security The fewer directories you keep in **PATH** when you are working with **root** privileges, the less likely you will be to execute an untrusted program while working with **root** privileges. If possible, keep only the default directories, along with **/usr/bin** and **/usr/sbin**, in **root**'s **PATH**. *Never include the working directory in **PATH** (as . or : : anywhere in **PATH**, or : as the last element of **PATH**).* For more information refer to "PATH: Where the Shell Looks for Programs" on page 359.

LE+ USING sudo TO GAIN root PRIVILEGES

If sudo (www.sudo.ws) is not set up so you can use it, and a **root** password exists and you know what it is, see "Administrator and the **wheel** group" on the next page for instructions on setting up sudo so you can use it to gain **root** privileges.

ADVANTAGES OF sudo

Using sudo rather than the **root** account for system administration offers many advantages.

- When you run sudo, it requests *your* password—not the **root** password—so you have to remember only one password.
- The sudo utility logs all commands it executes. This log can be useful for retracing your steps for system auditing and if you make a mistake.
- The sudo utility logs the username of a user who issues an sudo command. On systems with more than one administrator, this log tells you which users have issued sudo commands. Without sudo, you would not know which user issued a command while working with **root** privileges.
- The sudo utility allows implementation of a finer-grained security policy than does the use of su and the **root** account. Using sudo, you can enable specific users to execute specific commands—something you cannot do with the classic **root** account setup.
- Using sudo makes it harder for a malicious user to gain access to a system. When there is an unlocked **root** account, a malicious user knows the username of the account she wants to crack before she starts. When the **root** account is locked, the user has to determine the username *and* the password to break into a system.

- Managing **root** passwords over many systems is challenging. Remembering each system's password is difficult without writing them down (and storing them in a safe), and then retrieving a password is time-consuming. Keeping track of which users know which **root** passwords is impossible. Using sudo, even for full **root**-shell access, makes the tasks of gaining **root** privileges on a large number of systems and tracking who can gain **root** privileges on each system much easier.

SECURITY OF sudo

Some users question whether sudo is less secure than su. Because both rely on passwords, they share the same strengths and weaknesses. If the password is compromised, the system is compromised. However, if the password of a user who is allowed by sudo to do one task is compromised, the entire system might not be at risk. Thus, *if used properly,* the finer granularity of sudo's permissions structure *can* make it a more secure tool than su. Also, when sudo is used to invoke a single command, it is less likely that a user will be tempted to keep working with **root** privileges than if the user opens a **root** shell using su.

Using sudo might not always be the best, most secure way to set up a system. On a system used by a single user, there is not much difference between using sudo and carefully using su and a **root** password. In contrast, on a system with several users, and especially on a network of systems with central administration, sudo can be set up to be more secure than su.

USING sudo

Administrator and the **wheel** group

When you add or modify a user, you can specify that the user is an administrator by making the user a member of the **wheel** group. When you specify an account type of Administrator in the Users window (Figure 4-17, page 112), the user becomes a member of the **wheel** group. Based on the following line in the **/etc/sudoers** file, members of the **wheel** group can use sudo to gain **root** privileges:

```
%wheel  ALL=(ALL)       ALL
```

To make a user a member of the **wheel** group, working with **root** privileges, run usermod with the **–a** and **–G wheel** options to add the user to the **wheel** group. Substitute a username for **sam** in the following example.

```
# usermod -a -G wheel sam
# grep wheel /etc/group
wheel:x:10:root,sam
```

The grep command shows Sam as a member of the **wheel** group. See **/etc/group** on page 506 for more information on groups.

Timestamp

By default, sudo asks for *your* password (not the **root** password) the first time you run it. At that time, sudo sets your *timestamp*. After you supply a password, sudo will not prompt you again for a password for five minutes (by default), based on your timestamp.

EXECUTING A SINGLE COMMAND

In the following example, Sam tries to set the system clock while working as himself, a nonprivileged user. The date utility displays an error message followed by the expanded version of the date Sam entered. When he uses sudo to run date to set the system clock, sudo prompts him for his password, and the command succeeds.

```
$ date 12121500
date: cannot set date: Operation not permitted
Thu Dec 12 15:00:00 PST 2013

$ sudo date 12121500
[sudo] password for sam:
Thu Dec 12 15:00:00 PST 2013
```

Next Sam uses sudo to unmount a filesystem. Because he gives this command within five minutes of the previous sudo command, he does not need to supply a password:

```
$ sudo umount /music
$
```

Now Sam uses the **–l** option to check which commands sudo will allow him to run. Because the **sudoers** file is set up as explained in "Administrator and the **wheel** group," on the previous page, he is allowed to run any command as any user.

```
$ sudo -l
...
User sam may run the following commands on this host:
    (ALL) ALL
```

Granting root privileges to edit a file

tip With the **–e** option, or when called as sudoedit, sudo edits with **root** privileges the file named by its argument. By default sudo uses the vi editor; see page 432 for instructions on how to specify a different editor.

Any user who can run commands with **root** privileges can use the **–e** option. To give other users permission to edit any file using **root** privileges, specify in the **sudoers** file that that user can execute the command **sudoedit**. For more information refer to "User Privilege Specifications" on page 433.

Calling an editor in this manner runs the editor in the user's environment, maintaining the concept of least privilege. The sudo utility first copies the file to be edited to a temporary file that is owned by the user. If the file does not exist, sudo creates a new file that is owned by the user. Once the user has finished editing the file, sudo copies it back in place of the original file (maintaining the original permissions).

SPAWNING A root SHELL

When you have several commands you need to run with **root** privileges, it might be easier to spawn a **root** shell, give the commands without having to type **sudo** in front of each one, and exit from the shell. This technique defeats some of the safeguards built into sudo, so use it carefully and remember to return to a non**root** shell as soon

as possible. See the caution on least privilege on page 423. Use the sudo **–i** option (page 432) to spawn a **root** shell:

```
$ pwd
/home/sam
$ sudo -i
# id
uid=0(root) gid=0(root) groups=0(root),1(bin),2(daemon),3(sys) ...
# pwd
/root
# exit
logout
$
```

In this example, sudo spawns a **root** shell, which displays a **#** prompt to remind Sam that he is running with **root** privileges. The id utility displays the effective UID of the user running the shell. The exit command (you can also use CONTROL-D) terminates the **root** shell, returning Sam to his normal status and his former shell and prompt.

sudo's environment

The pwd builtin in the preceding example shows one aspect of the modified environment created by the **–i** option. This option spawns a **root** login shell (a shell with the same environment as a user logging in as **root** would have) and executes **root's** startup files (page 329). Before issuing the **sudo –i** command, the pwd builtin shows **/home/sam** as Sam's working directory; after the command, it shows **/root**, **root**'s home directory, as the working directory. Use the **–s** option (page 432) to spawn a **root** shell without modifying the environment. When you call sudo without an option, it runs the command you specify in an unmodified environment. To demonstrate this feature, the following example calls sudo without an option to run pwd. The working directory of a command run in this manner does not change.

```
$ pwd
/home/sam
$ sudo pwd
/home/sam
```

Redirecting output

The following command fails because although the shell that sudo spawns executes ls with **root** privileges, the nonprivileged shell that the user is running redirects the output. The user's shell does not have permission to write to **/root**.

```
$ sudo ls > /root/ls.sam
-bash: /root/ls.sam: Permission denied
```

There are several ways around this problem. The easiest is to pass the whole command line to a shell running under sudo:

```
$ sudo bash -c "ls > /root/ls.sam"
```

The bash **–c** option spawns a shell that executes the string following the option and then terminates. The sudo utility runs the spawned shell with **root** privileges. You must quote the string to prevent the nonprivileged shell from interpreting special characters. You can also spawn a **root** shell using **sudo –i**, execute the command, and exit from the privileged shell. (See the preceding section.)

optional Another way to deal with the issue of redirecting output of a command run by sudo is to use tee (page 162):

```
$ ls | sudo tee /root/ls.sam
...
```

This command line sends standard output of ls through sudo to tee, which copies its standard input to the file named by its argument and to standard output. If you do not want to display the output, you can have the nonprivileged shell redirect the output to **/dev/null** (page 503). The next example uses this technique to do away with the displayed output and uses the **–a** option to tee to append to the file instead of overwriting it:

```
$ ls | sudo tee -a /root/ls.sam > /dev/null
```

OPTIONS

You can use command-line options to control how sudo runs a command. Following is the syntax of an sudo command line:

*sudo [**options**] [**command**]*

where ***options*** is one or more options and ***command*** is the command you want to execute. Without the **–u** option, sudo runs ***command*** with **root** privileges. Some of the more common ***options*** follow; see the sudo man page for a complete list.

–b (**background**) Runs ***command*** in the background.

–e (**edit**) With this option, ***command*** is a filename and not a command. This option causes sudo to edit the file named ***command*** with **root** privileges using the editor named by the **SUDO_EDITOR**, **VISUAL**, or **EDITOR** environment variable. Default is the vi editor. Alternately, you can use the sudoedit utility without any options. Using this technique, the editor does not run with **root** privileges. See the tip on page 430.

–i (**initial login environment**) Spawns the shell that is specified for **root** (or another user specified by **–u**) in **/etc/passwd**, running **root**'s (or the other user's) startup files, with some exceptions (e.g., **TERM** is not changed). Does not take a ***command***. See "Spawning a **root** Shell" on page 426 for an example.

–k (**kill**) Resets the timestamp (page 429) of the user running the command, which means the user must enter her password the next time she runs sudo.

–L (**list defaults**) Lists the parameters that you can set on a Defaults line (page 436) in the **sudoers** file. Does not take a ***command***.

–l (**list commands**) Lists the commands the user who is running sudo is allowed to run on the local system. Does not take a ***command***.

–s (**shell**) Spawns a new **root** (or another user specified by **–u**) shell as specified in the **/etc/passwd** file. Similar to **–i** but does not change the environment. Does not take a ***command***.

–u *user* Runs ***command*** with the privileges of ***user***. Without this option, sudo runs ***command*** with **root** privileges.

LPI sudoers: CONFIGURING sudo

As installed, sudo is not as secure and robust as it can be if you configure it carefully. The sudo configuration file is **/etc/sudoers**. You can edit this file to give specific users the ability to run only certain commands with **root** privileges as opposed to any commands. You can also limit these users to running commands only with certain options or arguments. Or you can set up sudo so a specific user cannot use a specific option or argument when running a command with **root** privileges.

The best way to edit **sudoers** is to use visudo by giving the command: **su -c visudo** or **sudo visudo**. The visudo utility locks, edits, and checks the grammar of the **sudoers** file. By default, visudo calls the vi editor. You can set the **SUDO_EDITOR**, **VISUAL**, or **EDITOR** environment variable to cause visudo to call a different editor. The following command causes visudo to use the nano editor (page 270):

```
$ export EDITOR=$(which nano)
```

Replace **nano** with the textual editor of your choice. Put this command in a startup file (page 329) to set this variable each time you log in.

Always use visudo to edit the sudoers file

caution A syntax error in the **sudoers** file can prevent you from using sudo to gain **root** privileges. If you edit this file directly (without using visudo), you will not know that you introduced a syntax error until you find that you cannot use sudo. The visudo utility checks the syntax of **sudoers** before it allows you to exit. If it finds an error, it gives you the choice of fixing the error, exiting without saving the changes to the file, or saving the changes and exiting. The last choice is usually a poor one, so visudo marks it with **(DANGER!)**.

In the **sudoers** file, comments, which start with a hashmark (**#**), can appear anywhere on a line. In addition to comments, this file holds three types of entries: user privilege specifications, aliases, and defaults. Each of these entries occupies a line. You can continue a line by terminating it with a backslash (\).

USER PRIVILEGE SPECIFICATIONS

The format of a line that specifies user privileges is as follows (the whitespace around the equal sign is optional). In each case,

user_list host_list = [(runas_list)] command_list

- The ***user_list*** specifies the user(s) this specification line applies to. This list can contain usernames, groups (prefixed with **%**), and user aliases (next section). You can use the builtin alias ALL to cause the line to apply to all users.
- The ***host_list*** specifies the host(s) this specification line applies to. This list can contain one or more hostnames, IP addresses, or host aliases (next section). You can use the builtin alias ALL to cause the line to apply to all systems that refer to this **sudoers** file.
- The ***runas_list*** specifies the user(s) the commands in the ***command_list*** can be run as when sudo is called with the **–u** option (page 432). This list

can contain usernames, groups (prefixed with **%**), and runas aliases (next section). It must be enclosed within parentheses. Without ***runas_list***, sudo assumes **root**. You can use the builtin alias ALL to cause the line to apply to all usernames and groups.

- The ***command_list*** specifies the utilities this specification line applies to. This comma-separated list can contain names of utilities, names of directories holding utilities, and command aliases (next section). All names must be absolute pathnames; directory names must end with a slash (**/**). Precede a command with an exclamation point (**!**) to exclude that command. You can use the builtin alias ALL to cause the line to apply to all commands.

  Including the string **sudoedit** in the ***command_list*** gives users in the ***user_list*** permission to edit files using **root** privileges. See the tip on page 430 for more information.

If you follow a command in the ***command_list*** with two adjacent double quotation marks (" "), the user will not be able to specify any command-line arguments, including options to that command. Alternately, you can specify arguments, including wildcards, to limit the arguments a user is allowed to specify with that command.

Examples The following user privilege specification allows Sam to use sudo to mount and unmount filesystems (run mount and umount with **root** privileges) on all systems (as specified by **ALL**) that refer to the **sudoers** file containing this specification:

```
sam     ALL=(root) /bin/mount, /bin/umount
```

The (**root**) ***runas_list*** is optional. If you omit it, sudo allows the user to run the commands in the ***command_list*** with **root** privileges. In the following example, Sam takes advantage of these permissions. He cannot run umount directly; instead, he must call sudo to run it.

```
$ whoami
sam
$ umount /music
umount: only root can unmount /dev/sdb7 from /music
$ sudo umount /music
[sudo] password for sam:
$
```

If you replace the line in **sudoers** described above with the following line, Sam is not allowed to unmount **/p03**, although he can still unmount any other filesystem and can mount any filesystem:

```
sam     ALL=(root) /bin/mount, /bin/umount, !/bin/umount /p03
```

The result of the preceding line in **sudoers** is shown next. The sudo utility does not prompt for a password because Sam has entered his password within the last five minutes.

```
$ sudo umount /p03
Sorry, user sam is not allowed to execute '/bin/umount /p03' as root on localhost.
```

The following line limits Sam to mounting and unmounting filesystems mounted on **/p01**, **/p02**, **/p03**, and **/p04**:

```
sam     ALL= /bin/mount /p0[1-4], /bin/umount /p0[1-4]
```

The following commands show the result:

```
$ sudo umount /music
Sorry, user sam is not allowed to execute '/bin/umount /music' as root on localhost.
$ sudo umount /p03
$
```

Administrator: using the **wheel** group

As explained under "Administrator and the **wheel** group" on page 429, the following lines in **sudoers** allow users who are members of the **wheel** group (administrators) to use sudo to gain **root** privileges:

```
## Allows people in group wheel to run all commands
%wheel  ALL=(ALL)       ALL
```

This user privilege specification applies to all systems (as indicated by the **ALL** to the left of the equal sign). As the comment indicates, this line allows members of the **wheel** group (specified by preceding the name of the group with a percent sign: **%wheel**) to run any command (the rightmost **ALL**) as any user (the **ALL** within parentheses). When you call it without the **–u** option, the sudo utility runs the command you specify with **root** privileges, which is what sudo is used for most of the time.

If you modified the preceding line in **sudoers** as follows, it would allow members of the **wheel** group to run any command as any user with one exception: They would not be allowed to run passwd to change the **root** password (although they could gain **root** privileges and edit it manually).

```
%wheel ALL=(ALL) ALL, !/usr/bin/passwd root
```

optional

In the **%wheel ALL=(ALL) ALL** line in **/etc/sudoers**, if you replaced (**ALL**) with (**root**) or if you omitted (**ALL**), you would still be able to run any command with **root** privileges. You would not, however, be able to use the **–u** option to run a command as another user. Typically, when you can have **root** privileges, this limitation is not an issue. Working as a user other than **root** allows you to use the least privilege possible to accomplish a task, which is a good idea.

For example, if you are in the **wheel** group, the default entry in the **sudoers** file allows you to give the following command to create and edit a file in Sam's home directory. Because you are working as Sam, he will own the file and be able to read from and write to it.

```
$ sudo -u sam vi ~sam/reminder
$ ls -l ~sam/reminder
-rw-r--r--. 1 sam pubs 15 03-09 15:29 /home/sam/reminder
```

ALIASES

An alias enables you to rename and/or group users, hosts, or commands. Following is the format of an alias definition:

alias_type alias_name = *alias_list*

where ***alias_type*** is the type of alias (**User_Alias**, **Runas_Alias**, **Host_Alias**, **Cmnd_Alias**), ***alias_name*** is the name of the alias (by convention all uppercase), and ***alias_list*** is a comma-separated list of one or more elements that make up the alias. Preceding an element of an alias with an exclamation point (**!**) negates it.

User_Alias The ***alias_list*** for a user alias is the same as the ***user_list*** for a user privilege specification (previous section). The following lines from a **sudoers** file define three user aliases: OFFICE, ADMIN, and ADMIN2. The ***alias_list*** that defines the first alias includes the usernames **zach**, **sam**, and **sls**; the second includes two usernames and members of the **admin** group; and the third includes all members of the **admin** group except Sam.

```
User_Alias      OFFICE = zach, sam, sls
User_Alias      ADMIN = max, zach, %admin
User_Alias      ADMIN2 = %admin, !sam
```

Runas_Alias The ***alias_list*** for a runas alias is the same as the ***runas_list*** for a user privilege specification (previous section). The following SM runas alias includes the usernames **sam** and **sls**:

```
Runas_Alias     SM = sam, sls
```

Host_Alias Host aliases are meaningful only when the **sudoers** file is referenced by sudo running on more than one system. The ***alias_list*** for a host alias is the same as the ***host_list*** for a user privilege specification (previous section). The following line defines the LCL alias to include the systems named **guava** and **plum**:

```
Host_Alias      LCL = guava, plum
```

If you want to use fully qualified hostnames (**hosta.example.com** instead of just **hosta**) in this list, you must set the **fqdn** flag (next section). However, doing so might slow the performance of sudo.

Cmnd_Alias The ***alias_list*** for a command alias is the same as the ***command_list*** for a user privilege specification (previous section). The following command alias includes three files and by including a directory (denoted by its trailing **/**), incorporates all the files in that directory:

```
Cmnd_Alias      BASIC = /bin/cat, /usr/bin/vi, /bin/df, /usr/local/safe/
```

DEFAULTS (OPTIONS)

You can change configuration options from their default values by using the **Defaults** keyword. Most values in this list are flags that are implicitly Boolean (can either be on or off) or strings. You turn on a flag by naming it on a Defaults line, and you turn it off by preceding it with a **!**. The following line in the **sudoers** file turns off the **lecture** and **fqdn** flags and turns on **tty_tickets**:

```
Defaults        !lecture,tty_tickets,!fqdn
```

This section lists some common flags; see the **sudoers** man page for a complete list.

env_reset Causes sudo to reset the environment variables to contain the **LOGNAME**, **SHELL**, **USER**, **USERNAME**, **MAIL**, and **SUDO_*** variables only. Default is **on**. See the **sudoers** man page for more information.

fqdn (**fully qualified domain name**) Performs DNS lookups on *FQDNs* (page 1250) in the **sudoers** file. When this flag is set, you can use FQDNs in the **sudoers** file, but doing so might negatively affect sudo's performance, especially if DNS is not working. When this flag is set, you must use the local host's official DNS name, not an alias. If hostname returns an FQDN, you do not need to set this flag. Default is **on**.

insults Displays mild, humorous insults when a user enters a wrong password. Default is **off**. See also **passwd_tries**.

lecture=*freq* Controls when sudo displays a reminder message before the password prompt. Possible values of ***freq*** are **never**, **once**, and **always**. Specifying **!lecture** is the same as specifying a ***freq*** of **never**. Default is **once**.

mail_always Sends email to the **mailto** user each time a user runs sudo. Default is **off**.

mail_badpass Sends email to the **mailto** user when a user enters an incorrect password while running sudo. Default is **off**.

mail_no_host Sends email to the **mailto** user when a user whose username is in the **sudoers** file but who does not have permission to run commands on the local host runs sudo. Default is **off**.

mail_no_perms Sends email to the **mailto** user when a user whose username is in the **sudoers** file but who does not have permission to run the requested command runs sudo. Default is **off**.

mail_no_user Sends email to the **mailto** user when a user whose username is not in the **sudoers** file runs sudo. Default is **on**.

mailsub=*subj* (**mail subject**) Changes the default email subject for warning and error messages from the default ***** SECURITY information for %h ***** to ***subj***. The sudo utility expands **%h** within ***subj*** to the local system's hostname. Place ***subj*** between quotation marks if it contains shell special characters.

mailto=*eadd* Sends sudo warning and error messages to ***eadd*** (an email address; default is **root**). Place ***eadd*** between quotation marks if it contains shell special characters.

passwd_timeout=*mins*
The ***mins*** is the number of minutes before a sudo password prompt times out. A value of 0 (zero) indicates the password does not time out. Default is **5**.

passwd_tries=*num*
The ***num*** is the number of times the user can enter an incorrect password in response to the sudo password prompt before sudo quits. Default is **3**. See also **insults** and **lecture**.

rootpw Causes sudo to accept only the **root** password in response to its prompt. Because sudo issues the same prompt whether it is asking for your password or the **root** password, turning this flag on might confuse users. Default is **off**, causing sudo to accept the password of the user running sudo.

shell_noargs Causes sudo, when called without any arguments, to spawn a **root** shell without changing the environment. Default is **off**. This option causes the same behavior as the sudo **–s** option.

timestamp_timeout=*mins*
The ***mins*** is the number of minutes that the sudo timestamp (page 429) is valid. Set ***mins*** to **–1** to cause the timestamp to be valid forever; set to 0 (zero) to cause sudo to always prompt for a password. Default is 5.

tty_tickets Causes sudo to authenticate users on a per-tty basis, not a per-user basis. Default is **on**.

umask=*val* The ***val*** is the umask (page 469) that sudo uses to run the command that the user specifies. Set ***val*** to **0777** to preserve the user's umask value. Default is **0022**.

LOCKING THE root ACCOUNT (REMOVING THE root PASSWORD)

If you decide you want to lock the **root** account, give the command **su –c 'passwd –l root'**. This command renders the encrypted password in **/etc/shadow** invalid by prepending two exclamation points (**!!**) to it. You can unlock the account again by removing the exclamation points or by giving the command shown in the following example.

Unlocking the **root** account

If you decide you want to unlock the **root** account after locking it, give the following command. This command assumes you can use sudo to gain **root** privileges and unlocks the **root** account by assigning a password to it:

```
$ sudo passwd root
[sudo] password for sam:
Changing password for user root.
New password:
Retype new password:
passwd: all authentication tokens updated successfully.
```